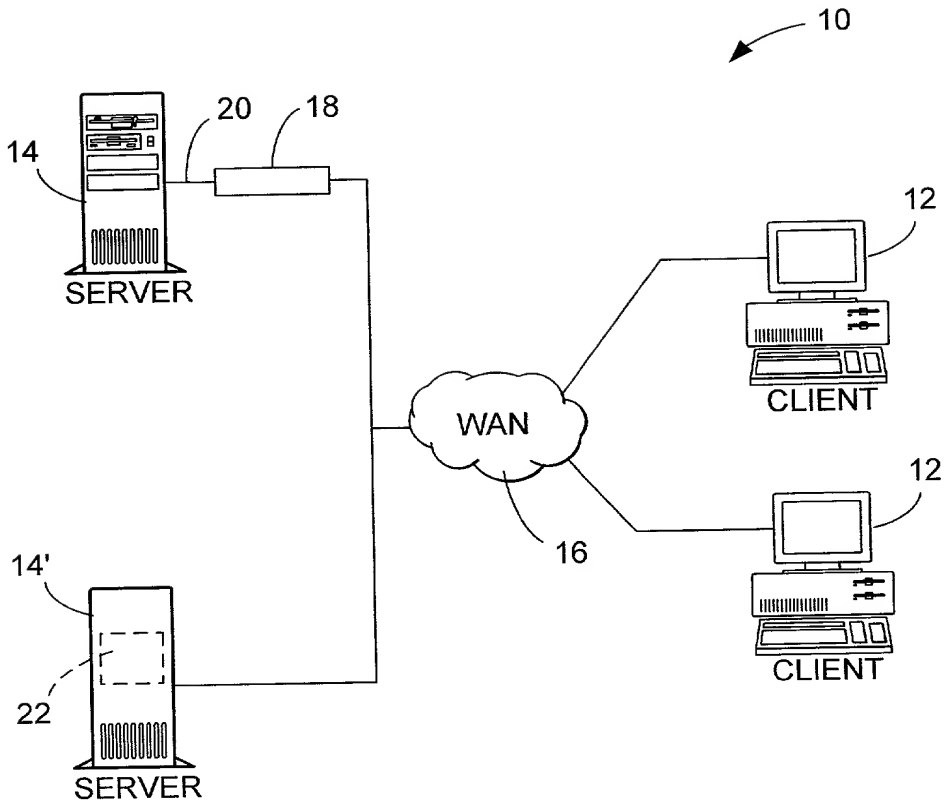

FIG. 1

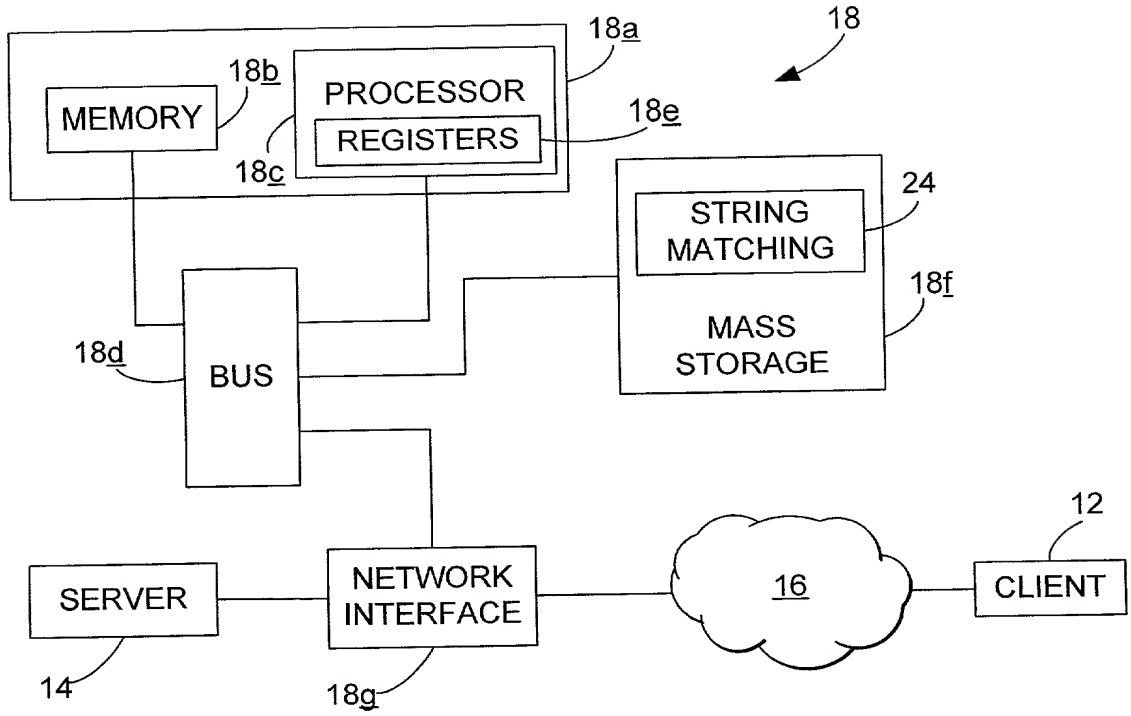
18
18a
18b
MEMORY
PROCESSOR
REGISTERS
18c
18e
24
STRING MATCHING
18f
MASS STORAGE
18d
BUS
12
SERVER
NETWORK INTERFACE
16
CLIENT
14
18g

FIG. 2

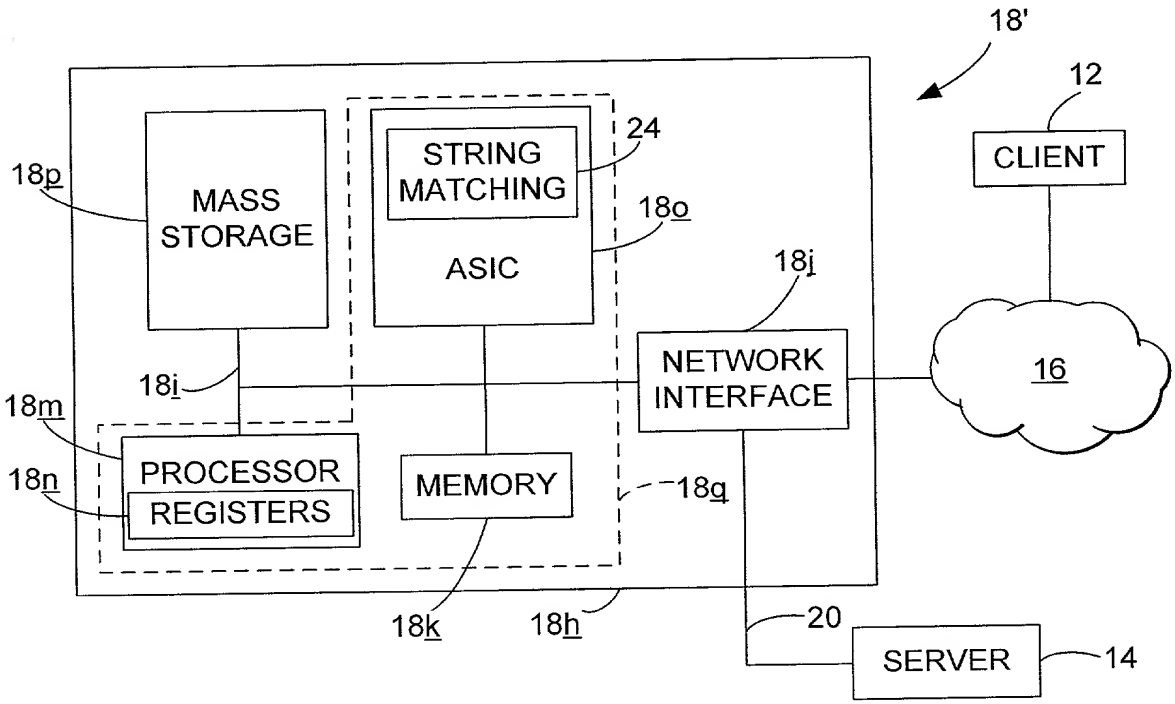
18'
12
CLIENT
18p
MASS STORAGE
24
STRING MATCHING
18o
ASIC
18j
NETWORK INTERFACE
16
18i
18m
18n
PROCESSOR
REGISTERS
MEMORY
18q
18k
18h
20
SERVER
14

FIG. 3

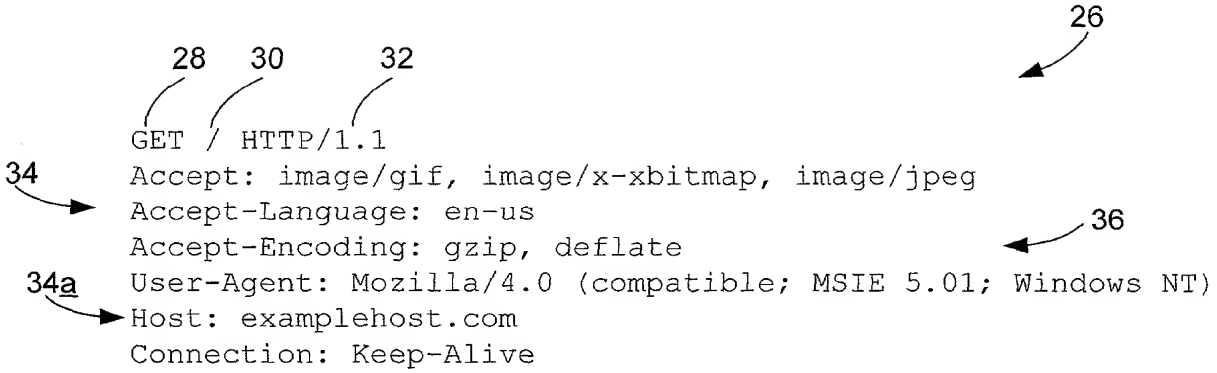
26
28
30
32
GET / HTTP/1.1
34
Accept: image/gif, image/x-xbitmap, image/jpeg
Accept-Language: en-us
36
Accept-Encoding: gzip, deflate
34a
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Host: examplehost.com
Connection: Keep-Alive

FIG. 4

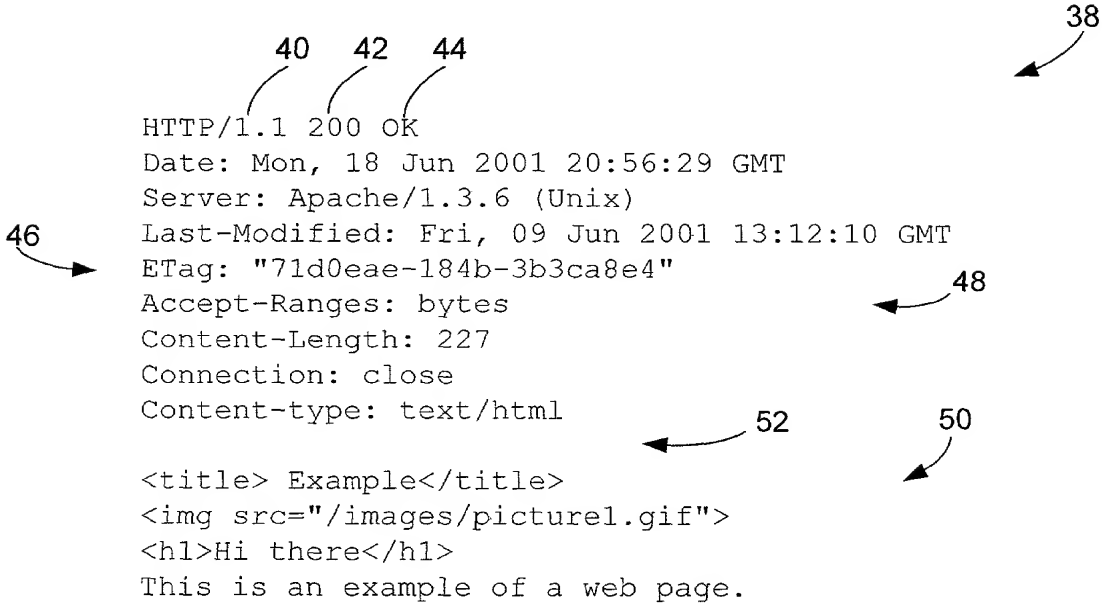
38
40
42
44
HTTP/1.1 200 OK
Date: Mon, 18 Jun 2001 20:56:29 GMT
Server: Apache/1.3.6 (Unix)
46
Last-Modified: Fri, 09 Jun 2001 13:12:10 GMT
ETag: "71d0eae-184b-3b3ca8e4"
48
Accept-Ranges: bytes
Content-Length: 227
Connection: close
Content-type: text/html
52
50
<title> Example</title>
<img src="/images/picture1.gif">
<h1>Hi there</h1>
This is an example of a web page.

FIG. 5

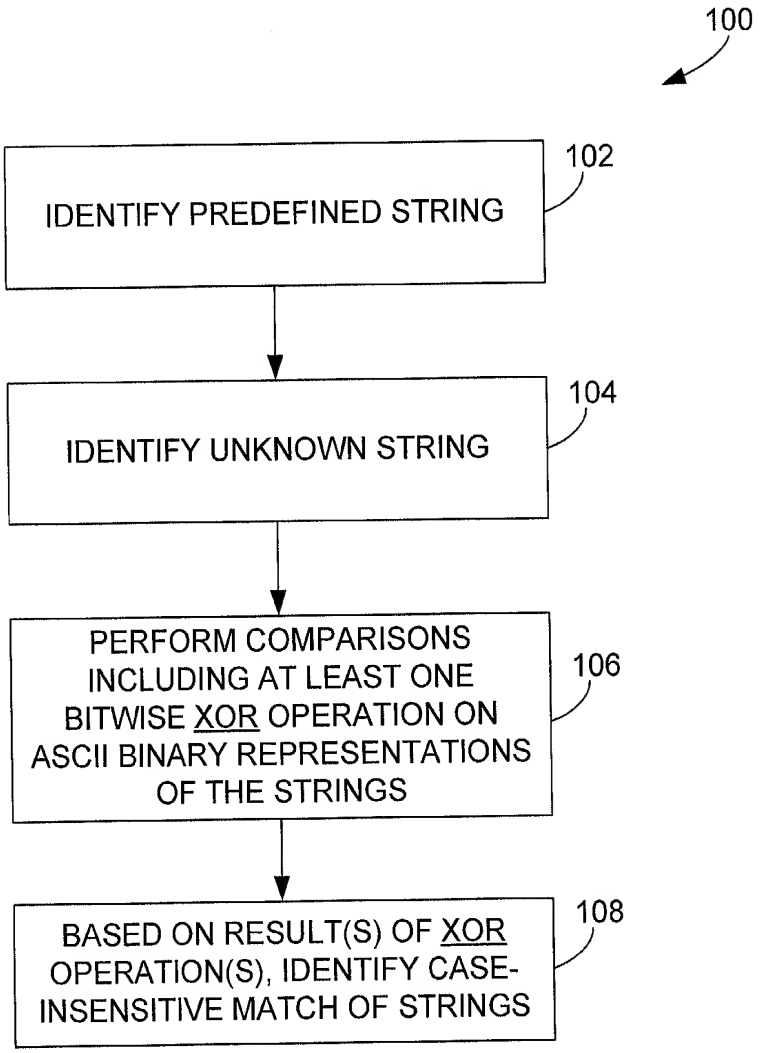
100
IDENTIFY PREDEFINED STRING
102
IDENTIFY UNKNOWN STRING
104
PERFORM COMPARISONS INCLUDING AT LEAST ONE BITWISE XOR OPERATION ON ASCII BINARY REPRESENTATIONS OF THE STRINGS
106
BASED ON RESULT(S) OF XOR OPERATION(S), IDENTIFY CASE-INSENSITIVE MATCH OF STRINGS
108

FIG. 6

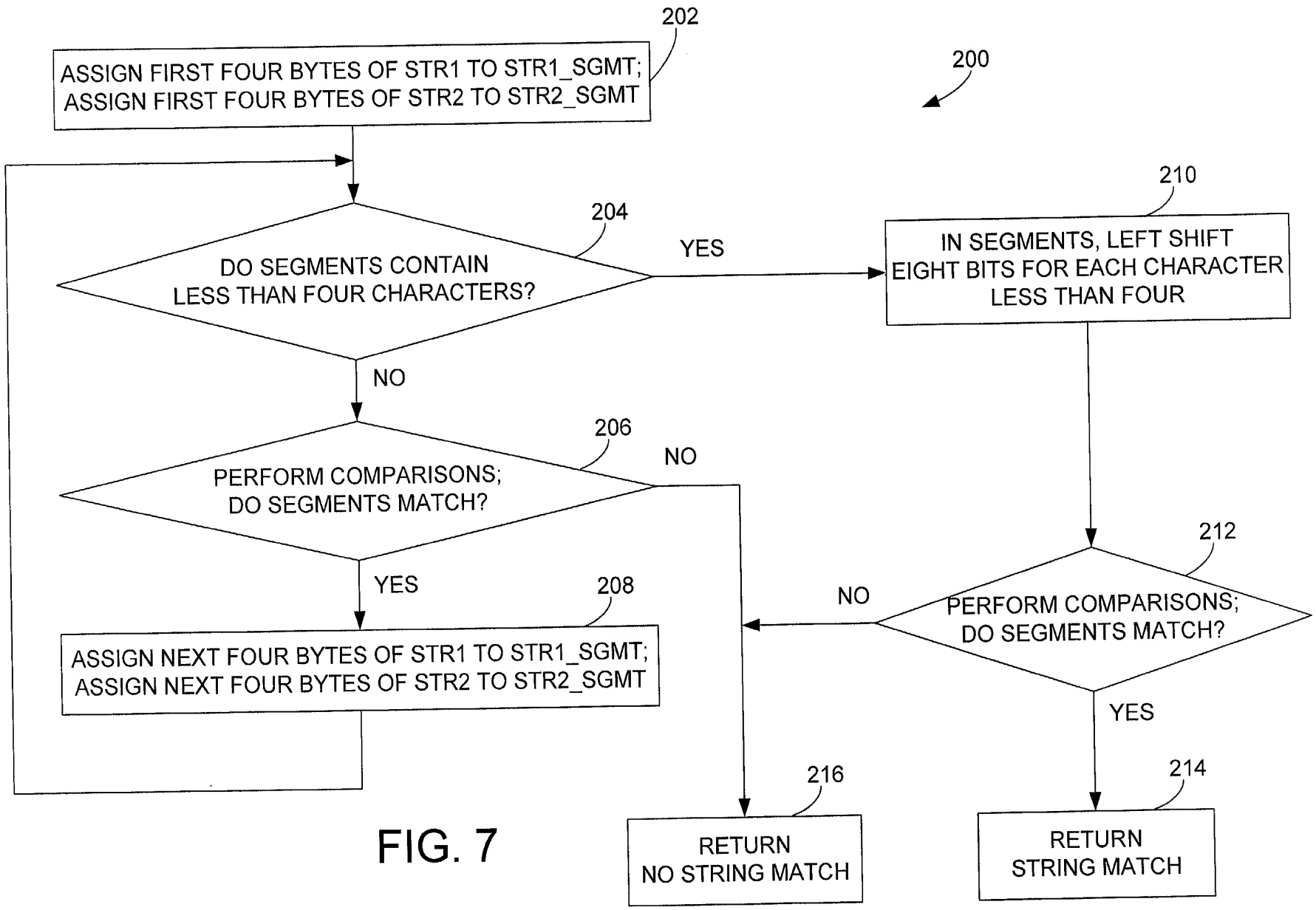
200
202
ASSIGN FIRST FOUR BYTES OF STR1 TO STR1_SGMT;
ASSIGN FIRST FOUR BYTES OF STR2 TO STR2_SGMT
204
DO SEGMENTS CONTAIN
LESS THAN FOUR CHARACTERS?
YES
NO
210
IN SEGMENTS, LEFT SHIFT
EIGHT BITS FOR EACH CHARACTER
LESS THAN FOUR
206
PERFORM COMPARISONS;
DO SEGMENTS MATCH?
NO
YES
208
ASSIGN NEXT FOUR BYTES OF STR1 TO STR1_SGMT;
ASSIGN NEXT FOUR BYTES OF STR2 TO STR2_SGMT
212
PERFORM COMPARISONS;
DO SEGMENTS MATCH?
NO
YES
216
RETURN
NO STRING MATCH
214
RETURN
STRING MATCH

FIG. 7

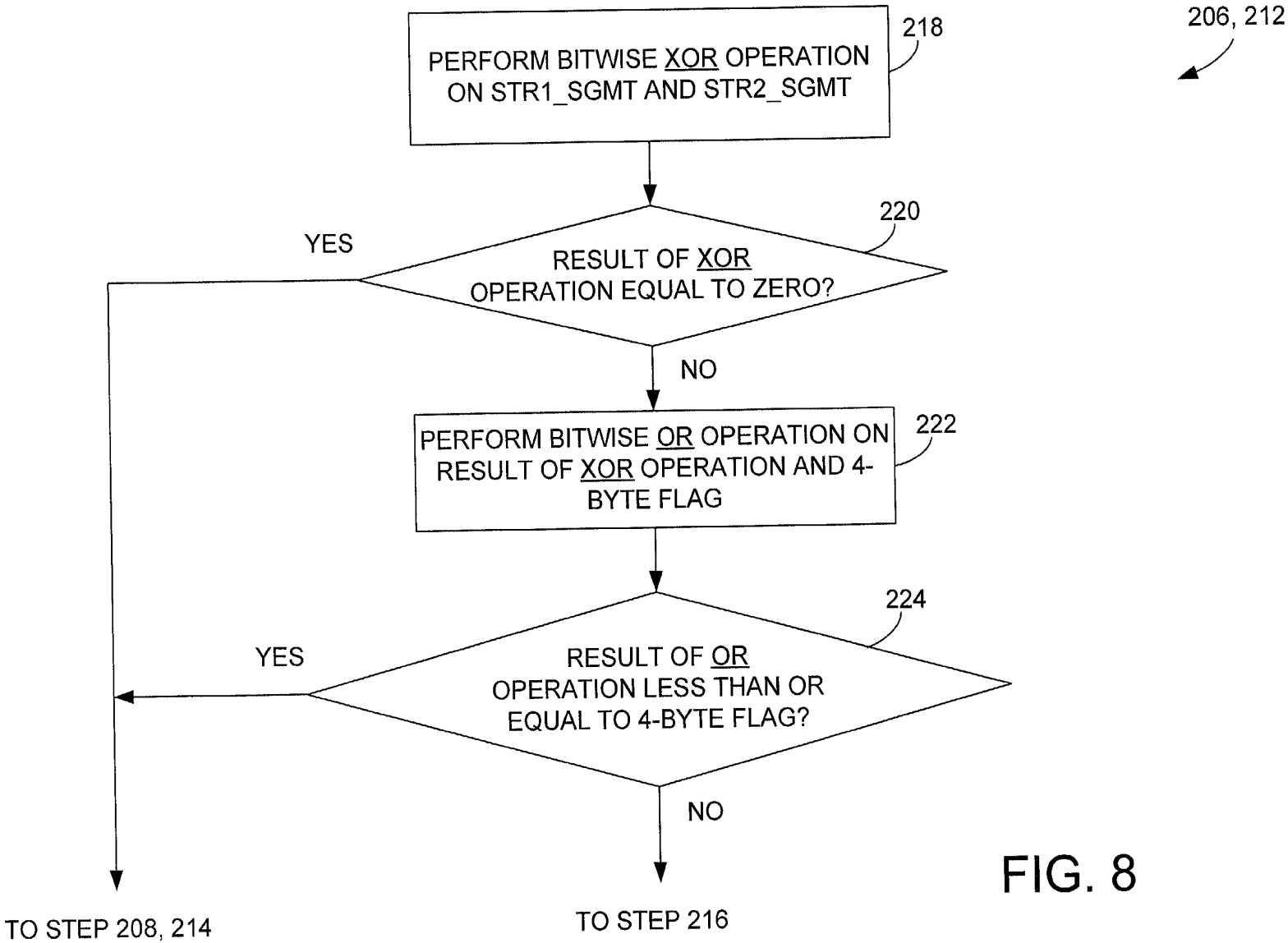
206, 212
PERFORM BITWISE XOR OPERATION ON STR1_SGMT AND STR2_SGMT
218
RESULT OF XOR OPERATION EQUAL TO ZERO?
220
YES
NO
PERFORM BITWISE OR OPERATION ON RESULT OF XOR OPERATION AND 4-BYTE FLAG
222
RESULT OF OR OPERATION LESS THAN OR EQUAL TO 4-BYTE FLAG?
224
YES
NO
TO STEP 208, 214
TO STEP 216

FIG. 8

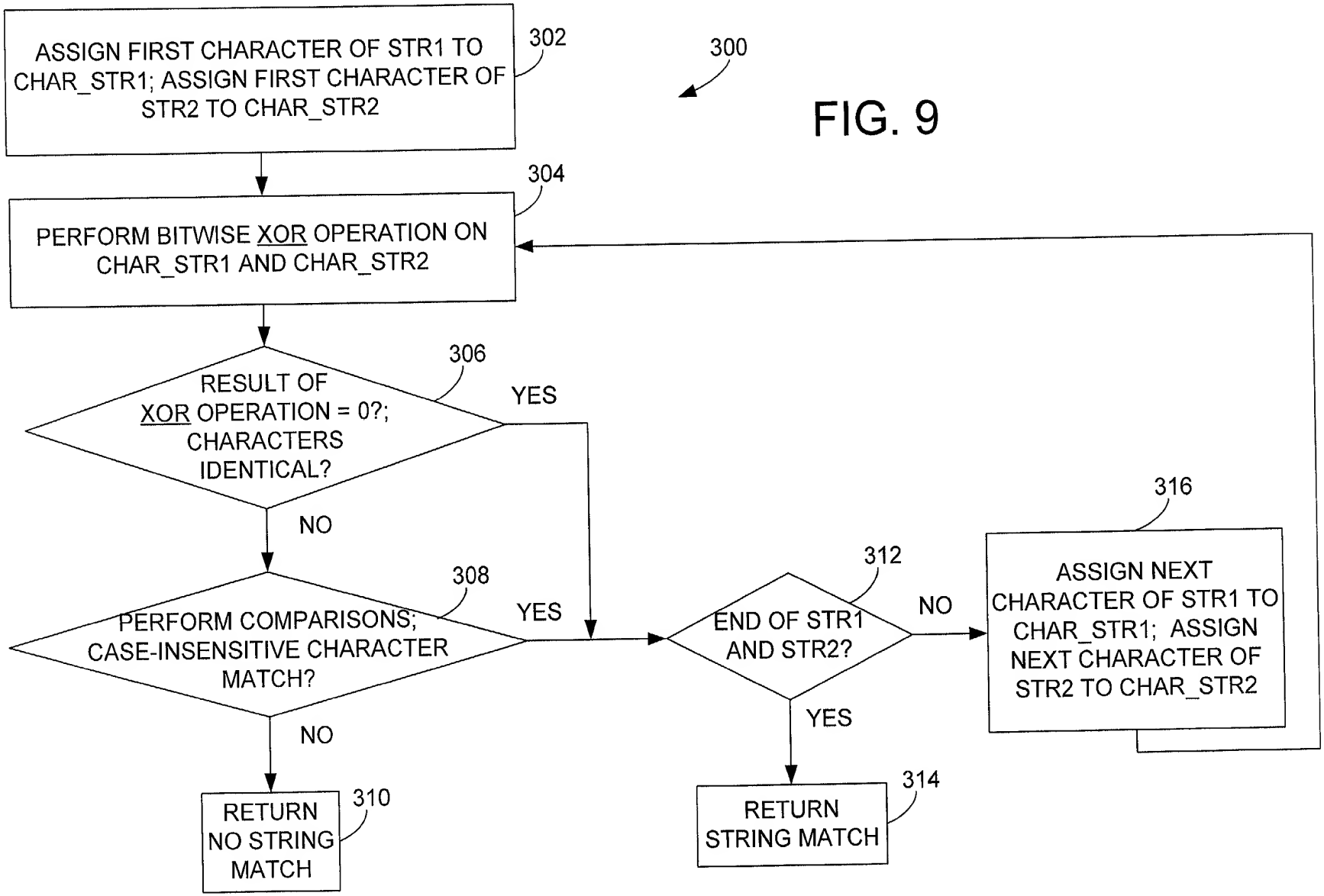
ASSIGN FIRST CHARACTER OF STR1 TO CHAR_STR1; ASSIGN FIRST CHARACTER OF STR2 TO CHAR_STR2
302
300
PERFORM BITWISE XOR OPERATION ON CHAR_STR1 AND CHAR_STR2
304
RESULT OF XOR OPERATION = 0?; CHARACTERS IDENTICAL?
306
YES
NO
PERFORM COMPARISONS; CASE-INSENSITIVE CHARACTER MATCH?
308
YES
NO
END OF STR1 AND STR2?
312
NO
YES
ASSIGN NEXT CHARACTER OF STR1 TO CHAR_STR1; ASSIGN NEXT CHARACTER OF STR2 TO CHAR_STR2
316
RETURN NO STRING MATCH
310
RETURN STRING MATCH
314

FIG. 9

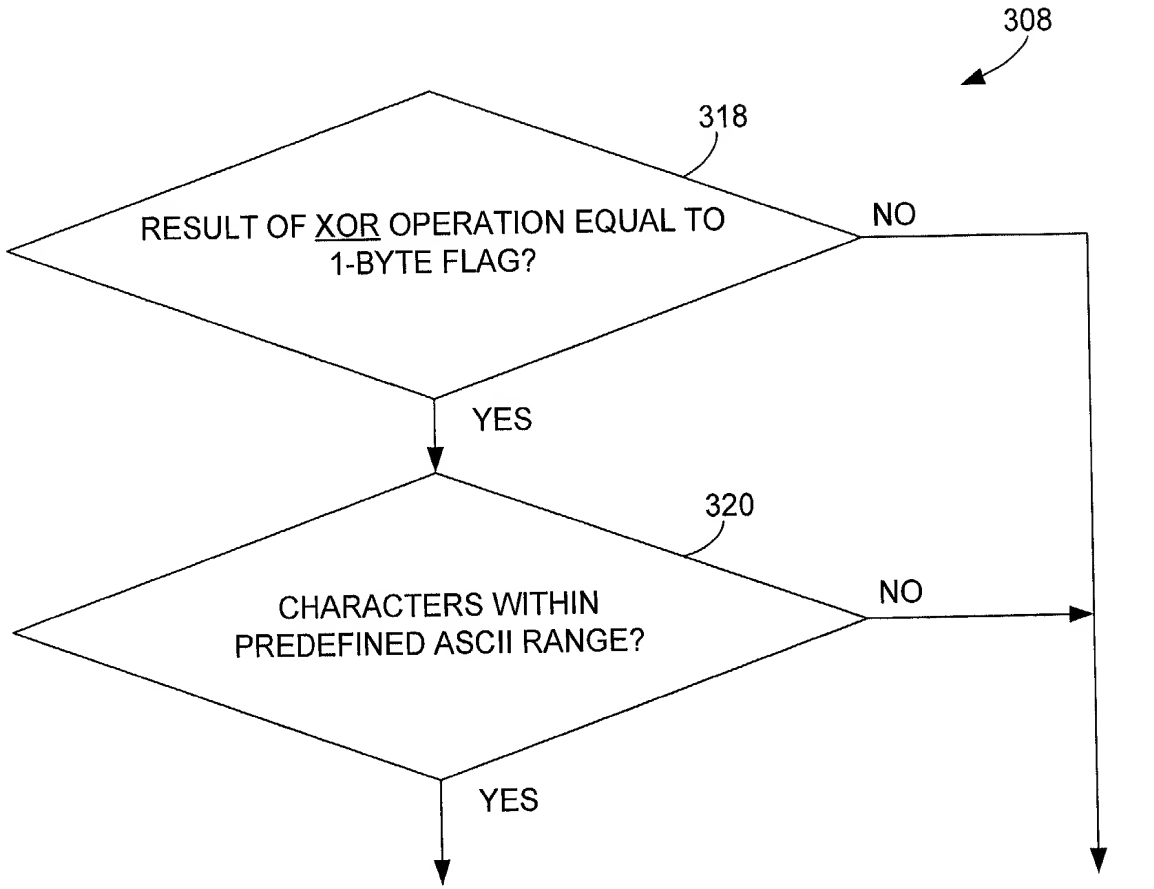
308
318
RESULT OF XOR OPERATION EQUAL TO 1-BYTE FLAG?
NO
YES
320
CHARACTERS WITHIN PREDEFINED ASCII RANGE?
NO
YES
TO STEP 312
TO STEP 310

FIG. 10

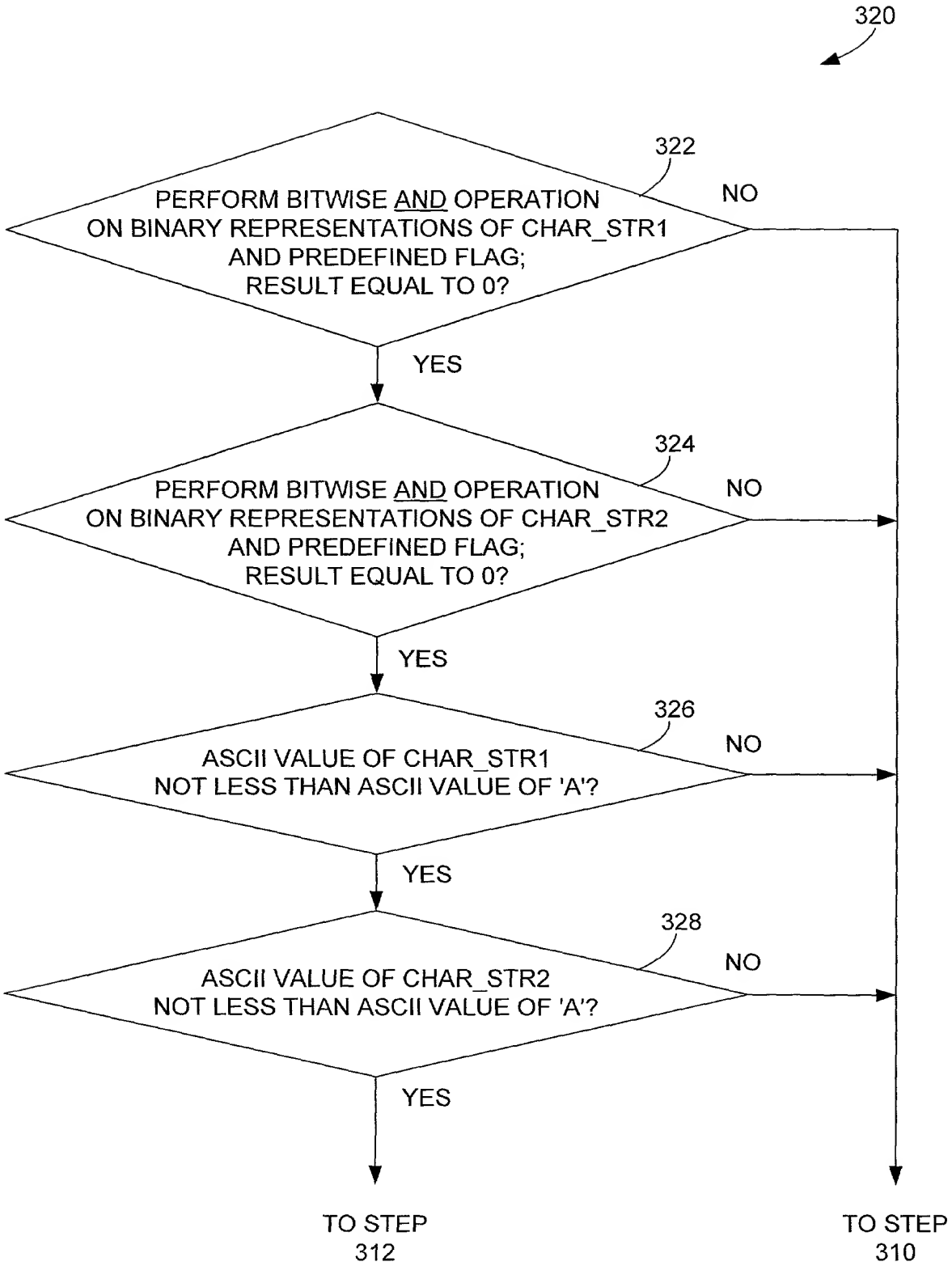
320
322
PERFORM BITWISE AND OPERATION ON BINARY REPRESENTATIONS OF CHAR_STR1 AND PREDEFINED FLAG; RESULT EQUAL TO 0?
NO
YES
324
PERFORM BITWISE AND OPERATION ON BINARY REPRESENTATIONS OF CHAR_STR2 AND PREDEFINED FLAG; RESULT EQUAL TO 0?
NO
YES
326
ASCII VALUE OF CHAR_STR1 NOT LESS THAN ASCII VALUE OF 'A'?
NO
YES
328
ASCII VALUE OF CHAR_STR2 NOT LESS THAN ASCII VALUE OF 'A'?
NO
YES
TO STEP 312
TO STEP 310

FIG. 11

Dec	Binary	Symbol
032	00100000	Space
033	00100001	!
034	00100010	"
035	00100011	#
036	00100100	$
037	00100101	%
038	00100110	&
039	00100111	'
040	00101000	(
041	00101001	)
042	00101010	*
043	00101011	+
044	00101100	,
045	00101101	-
046	00101110	.
047	00101111	/
048	00110000	0
049	00110001	1
050	00110010	2
051	00110011	3
052	00110100	4
053	00110101	5
054	00110110	6
055	00110111	7
056	00111000	8
057	00111001	9
058	00111010	:
059	00111011	;
060	00111100	<
061	00111101	=
062	00111110	>
063	00111111	?
064	01000000	@
065	01000001	A
066	01000010	B
067	01000011	C
068	01000100	D
069	01000101	E
070	01000110	F
071	01000111	G
072	01001000	H
073	01001001	I
074	01001010	J
075	01001011	K
076	01001100	L
077	01001101	M
078	01001110	N
079	01001111	O
080	01010000	P
081	01010001	Q
082	01010010	R
083	01010011	S
084	01010100	T
085	01010101	U
086	01010110	V
087	01010111	W
088	01011000	X
089	01011001	Y
090	01011010	Z
091	01011011	[
092	01011100	\
093	01011101	]
094	01011110	^
095	01011111	_
096	01100000	`
097	01100001	a
098	01100010	b
099	01100011	c
100	01100100	d
101	01100101	e
102	01100110	f
103	01100111	g
104	01101000	h
105	01101001	i
106	01101010	j
107	01101011	k
108	01101100	l
109	01101101	m
110	01101110	n
111	01101111	o
112	01110000	p
113	01110001	q
114	01110010	r
115	01110011	s
116	01110100	t
117	01110101	u
118	01110110	v
119	01110111	w
120	01111000	x
121	01111001	y
122	01111010	z
123	01111011	{
124	01111100	|
125	01111101	}
126	01111110	~
127	01111111	DEL

330

FIG. 12

```
int
rln_strncasematch_cheat(const char *str1, const char *str2,
                                register int len)
{
     u_int32_t *hold1, *hold2;
     register u_int32_t match;
     u_int32_t shold1, shold2;
     register int i;

     for (i=0; i <= len - 4; i += 4) {

          hold1 = (u_int32_t *)(str1 + i);
          hold2 = (u_int32_t *)(str2 + i);

          match = *hold1 ^ *hold2;

          if (match != 0 && ((match | LCASE_HIT) > LCASE_HIT)) {
               return 0;
          }
     }

     if (i < len) {
          hold1 = (u_int32_t *)(str1 + i);
          hold2 = (u_int32_t *)(str2 + i);

          shold1 = *hold1 << (4 - (len - i)) * 8;
          shold2 = *hold2 << (4 - (len - i)) * 8;

          match = shold1 ^ shold2;

          if (match != 0 && ((match | LCASE_HIT) > LCASE_HIT)) {
               return 0;
          }
     }

     return 1;
}
```

FIG. 13


```
int
rln_strncasematch(const register unsigned char *str1,
                  const register unsigned char *str2,
                  const register int len)
{
     register int i;
     register unsigned char match;

     for (i=0; i<len; i++) {

          match = str1[i] ^ str2[i];

          if (!match)
               continue;

          if (match != LCASE || (str1[i] & EIGHTBIT) ||
                    (str2[i] & EIGHTBIT) || (str1[i] < 'A')
                     || (str2[i] < 'A')) {
               return 0;
          }
     }
     return 1;
}
```

FIG. 14